Mining Top-K Largest Tiles in a Data Stream

Hoang Thanh Lam, Wenjie Pei, Adriana Prado, Baptiste Jeudy, Elisa Fromont

Mining Top-K Largest Tiles in a Data Stream

Hoang Thanh Lam[1,3], Wenjie Pei[1], Adriana Prado[2], Baptiste Jeudy[2], and Élisa Fromont[2]

[1] Technische Universiteit Eindhoven, 5612 AZ, Eindhoven, Netherlands
[2] H. Curien Lab, UMR CNRS 5516, Université de St-Etienne, Université de Lyon, France
[3] IBM Research Lab, Damastown, Dublin, Ireland

Abstract. Large tiles in a database are itemsets with the largest *area* which is defined as the itemset frequency in the database multiplied by its size. Mining these large tiles is an important pattern mining problem since tiles with a large area describe a large part of the database. In this paper, we introduce the problem of mining top-k largest tiles in a data stream under the sliding window model. We propose a candidate-based approach which summarizes the data stream and produces the top-k largest tiles efficiently for moderate window size. We also propose an approximation algorithm with theoretical bounds on the error rate to cope with large size windows. In the experiments with two real-life datasets, the approximation algorithm is up to hundred times faster than the candidate-based solution and the baseline algorithms based on the state-of-the-art solutions. We also investigate an application of large tile mining in computer vision and in emerging search topics monitoring.

1 Introduction

Mining frequent patterns is an important research topic in data mining. However, instead of focusing on exhaustive search to find all possible frequent patterns, many works are now focusing on designing methods that are not only efficient in the context of very big data but also limit, e.g. with constraints or with new interestingness measures, the number of patterns output by these algorithms.

Area is a measure of pattern interestingness defined as a pattern's frequency in the database multiplied by its size. It has been shown that in some applications such as in role mining [10, 16] where the idea is, given a set of users and a set of permissions, to find a minimum set of roles such that all users will be assigned a role for which some permissions will be granted, mining roles is equivalent to a variant of mining itemsets with large area in a database.

Recent applications produce a large variety of transactional data streams, such as text stream from *twitter*[4] or video stream in which video frames can be converted into transactions [6]. In the context of data streams, data usually arrive continuously with high speed, hence requiring efficient mining techniques for summarizing the data stream and keeping track of important patterns.

[4] `www.twitter.com`

In this paper we tackle the problem of mining the top-k largest tiles, i.e. the k closed itemsets with the largest area in a stream of itemsets. The problem of mining the largest tile in a database is well-known to be NP-hard and inapproximable [7]. Therefore, the straightforward approach that recalculates the set of top-k largest tiles from scratch every time a sliding window is updated is not efficient. To deal with this situation, we first introduce in Section 4 a candidate-based approach which incrementally summarizes the stream by keeping the itemsets that can be the top-k largest tiles in some future windows. For each candidate, an *upper-bound* and a *lower-bound* of the area in any future window are kept. These bounds are used to prune the candidates when they cannot be the top-k largest tiles in any future window. In doing so, the candidate-based algorithm is more efficient than the straightforward approach because updating is cheaper than recalculating the top-k tiles from scratch.

However, when the widow size is large, the candidate-based algorithm is inefficient because the summary grows very quickly. Therefore, we introduce an approximation algorithm with theoretical bounds on the error rate. In the experiments with two real-life datasets presented in Section 6, the approximation algorithm is two to three orders of magnitude faster and more accurate than the candidate-based solution and the baseline algorithms created based on the state-of-the-art solutions for the top-k largest tiles mining. We also discuss potential applications of mining large tiles for object tracking in video and emerging search topics monitoring problems.

2 Related Works

Recent works [12, 9, 17] propose approaches to solve the redundancy and trivial patterns issues in frequent pattern mining. For instance, the first two works focuse on finding a relevant or concise representation of sets of frequent patterns. Meanwhile, the last work solves the aforementioned issues by proposing a MDL-based approach that finds patterns compressing the database well. In all these cases as well as in ours, the purpose is to limit the output of the algorithms to a set of useful patterns.

The problem of mining large tiles has already been tackled in [10, 16, 13]. The authors of the two first papers show that the problem of finding roles is equivalent to variants of the tiling database problem. The last paper shows how tiles can be used to understand complex proteins by identifying their subunits. The authors of [4] also showed that the dense rectangles in a binary matrix seem to correspond to interesting concepts. Therefore, mining large tiles may help to identify those interesting concepts from the databases. In [15] the authors investigate how to output the set of tiles in a tree representation which is easily interpretable by the users. Tiles are also used to identify and characterize anomalies in a database in [14].

In many applications, data arrives in a streaming fashion with high speed [1]. Besides the popularity of data streams in many applications, the temporal aspect of the patterns such as the evolution of patterns overtime [1] provides

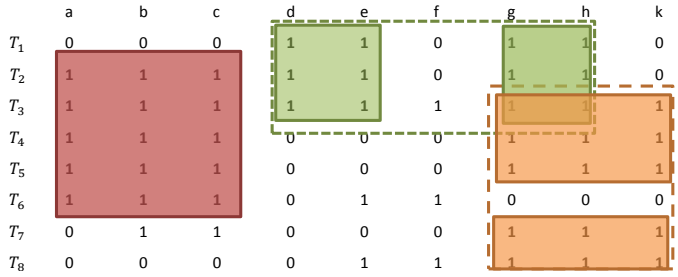


Fig. 1. An example of large tiles in a window with $w = 8$ transactions.

useful insights about the data for the users. Despite the importance of data stream paradigm, there is no work yet addressing the problem of mining large tiles in a data stream. The algorithms introduced in this paper are, to the best of our knowledge, the first to solve the problem of mining the top-k largest tiles in a stream of itemsets under the sliding windows model.

3 Problem Definition

Let Σ be an alphabet of items, a transaction T is an itemset $T \subseteq \Sigma$. Let $S = T_1 T_2 \cdots T_n$ be a stream of transactions where each transaction T_t is associated with a timestamp t indicating the order of transaction generation.

In this work we consider the sliding window model in which a window of the w most recent transactions is monitored. A sliding window of size w at time point t denoted by W_t is a sequence of w consecutive transactions in the data stream, i.e. $W_t = T_{t-w+1} \cdots T_{t-1} T_t$.

For any given itemset $I \subseteq \Sigma$ the frequency of I in a sliding window W_t, denoted by $f_t(I)$, is defined as the number of transactions in W_t that are the supersets of I. Let $|I|$ be the cardinality of the set I, the area of I in the window W_t denoted by $A_t(I)$ is defined as $A_t(I) = f_t(I) * |I|$.

An itemset I is closed in a window W_t if there is no superset of I with the same frequency as I in the window W_t. Such itemsets are usually called large tiles in the literature [7]. Itemsets that are not closed correspond to sub-tiles of large tiles and are thus not interesting for our problem. From now on we use the term tiles to refer to closed itemsets.

Example 1 (Large tiles). Figure 1 shows a window with 8 transactions represented as rows of a binary matrix. In each row, an element is equal to 1 if the corresponding item belongs to the transaction. In this figure, three large tiles *abc*, *degh* and *ghk* with respective area 15, 12 and 15 are highlighted. Large tiles are the maximal sub-matrices containing only 1.

The problem of stream tiling can be formulated as :

Definition 1 (Stream Tiling). *Given a data stream of itemset transactions S, a parameter k and a window size w, the stream tiling problem consists in computing the k tiles with the largest area in every window of size w.*

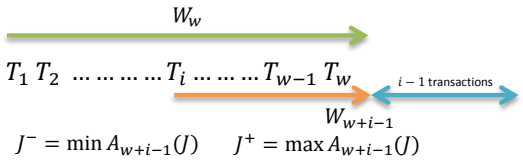
4 Algorithms

It was proven that the problem of mining the largest tile in a database is NP-Complete and is even inapproximable [7]. Therefore, recalculation of the top-k largest tiles from scratch every time the window is updated is very time-demanding. In this section we discuss efficient solutions for the given problem under the streaming context.

4.1 Candidate-Based Algorithm

We first discuss an exact algorithm named *cTile* which maintains a summary of the sliding window containing candidate itemsets that potentially can become top-k largest tiles in any future window. This summary is designed such that it can avoid expensive recalculation of the set of largest tiles from scratch when the window is sliding.

The general idea of the algorithm is as follows: at every transaction T_i we keep a candidate list C_i of all closed subsets of the given transaction which can become a top-k tile in a future sliding window. In order to identify these closed itemsets, we keep a *lower-bound* and an *upper-bound* on the area of these itemsets in any future sliding window which contains T_i. These bounds will be used to infer which itemsets can be top-k tiles and which sets for sure cannot be top-k tiles in any future window (thus can be removed from the summary).



Given a window $W_w = T_1 \cdots T_{w-1} T_w$, for every closed itemset J in the candidate list $C_i (1 \leq i \leq w)$, the *lower-bound* and the *upper-bound* of the area of this itemset in any future sliding window containing T_i are denoted J^- and J^+. The lower-bound J^- at time point w is calculated as the area of J in the transactions $T_i, T_{i+1}, \cdots, T_w$ and the *upper-bound* J^+ is calculated as

$$J^+ = J^- + (i-1) \times |J| \tag{1}$$

Proposition 1. *J^- and J^+ are the correct lower-bound and upper-bound on the area of J in any future window W_t of size w, $w \leq t$, containing the transaction T_i.*

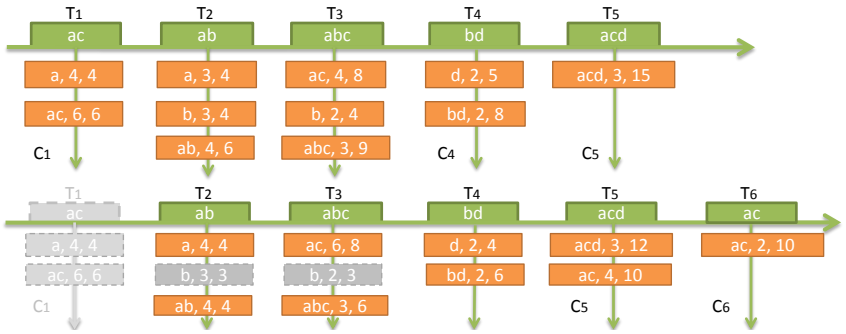


Fig. 2. An example of the summary maintained by the *cTile* algorithm for sliding windows with size $w = 5$. The summary contains candidate itemsets which can become a top-k tile in the future. Every candidate is associated with a *lower-bound* and an *upper-bound* on the area which are used to prune the candidate list.

Example 2 (Bounds). Figure 2 (upper-part) shows an example of a sliding window with size $w = 5$. Each candidate J in a candidate list C_i is associated with two numbers corresponding to the lower-bound and the upper-bound on the area of the candidate. For instance, for the candidate $J = ac$ associated with the candidate list C_3 the lower-bound is $J^- = 4$ because the area of ac in the set of currently observed transactions $T_3, \cdots, T_5$ is $2 * 2 = 4$. For any future window in which the transaction T_3 is not expired, the area of ac is at least as large as 4. Meanwhile $J^+ = 8$ because the area of ac in any future window that contains T_3 is at most $4 + 2 * 2 = 8$. The area is equal to J^+ only when the new transactions T_6 and T_7 both contain ac.

Algorithm 1 incrementally maintains the candidate lists C_i in the summary. When a new transaction is appended to the end of the window, the bounds J^+ and J^- are updated accordingly (line 10-13). Then the new transaction is intersected with every element in the summary and the intersections are added to the corresponding candidate list (line 8-9). If there exists another candidate $B \in C_j$ kept at a younger transaction T_j $(j > i)$, such that B^- is ranked k^{th} in all transactions younger than T_i (including itself) and meanwhile $B^- > J^+$, then J will never be among top-k tiles in any future sliding window, hence J is removed from the summary. When a transaction is expired, it is removed from the window along with the candidate list stored at the given transaction.

Example 3 (cTile). Figure 2 (bottom part) shows one update step of Algorithm 1 when a new transaction $T_6 = \{a, c\}$ is appended to the window and T_1 together with C_1 are removed from the summary. First, T_6 is intersected with all the existing candidates in $C_2, C_3, \cdots, C_5$ to add new closed sets to the candidate lists. After that all the lower-bounds and the upper-bounds are updated accordingly.

Assume that we want to get top-3 largest tiles in the sliding window, i.e. $k = 3$. Since $(b, 3, 3)$ in C_2 has the *upper-bound* equal to 3. It is removed from the summary because the itemset ranked at the third position in the candidates lists

Algorithm 1 cTile(S_t)

Input: A stream of transactions $T_1, T_2, \cdots, T_\infty$, a sliding window size w and a parameter k
Output: Summary C for calculating top-k largest tiles
$C \longleftarrow \{C_1, C_2, \cdots, C_w\}$ //candidate lists
for $t = w \rightarrow \infty$ **do**
  $C \leftarrow C \setminus \{C_{t-w+1}\}$
  $C_{t+1} \leftarrow \{T_{t+1}\}$
  $C \leftarrow C \cup \{C_{t+1}\}$
  for $i = t \rightarrow t - w + 2$ **do**
    $C_i \longleftarrow C_i \bigcup \text{intersect}(C_i, T_{t+1})$
    for $J \in C_i$ **do**
      $J^- = |\{i \leq l \leq t | J \subseteq T_l\}| * |J|$
      $J^+ = J^- + (w - t + i - 1)|J|$
    end for
    Pruning(C_i)
  end for
end for
Return C

of younger transactions (including T_2) are $(a, 4, 4)$, $(ac, 4, 10)$, $(ab, 4, 4)$ whose *lower-bound* is 4. The same pruning operation can be applied for $(b, 2, 3)$ in C_3.

Theorem 1 *Given k, w and a stream of transactions, using the summary in Algorithm 1 we can answer the top-*k *largest tiles exactly.*

Proof. Assume that there exists a top-k largest tile I in a window W_t not recognized as a top-k largest tile (false negative). The first transaction of W_t containing I is denoted by T_i. False negative can only occur if at time t, $I \notin C_i$. Therefore either I is directly pruned from C_i by pruning criteria or indirectly pruned because its closed supersets are pruned.

Since the bounds are exact the former case cannot happen. The latter case happens when I is not a closed itemset at the moment its closed supersets are pruned. In such case, the upper-bound of I is always less than the upper-bound of its closed supersets which were used to prune the supersets. Therefore, I is not a top-k tile in W_t. Both cases lead to contradiction.

4.2 Approximation Algorithm

The size of the summary maintained by *cTile* grows quickly when the window size increases making it inefficient for monitoring large windows. The main reason is that each time a new transaction arrives, it has to be intersected with a large amount of candidates, which is a time-consuming operation. Therefore, in this section we discuss an approximation algorithm named *aTile* that approximates the set of largest tiles efficiently.

The main process of the approximation algorithm is almost the same as Algorithm 1. The only difference of these two algorithms lies in the method

of candidate pruning. Instead of using an *upper-bound* on the area, *aTile* tries to approximate the future area of a tile. This estimate is used to prune the candidates in the same way as *cTile* does.

Let us denote the probability of observing an itemset J as a subset of a transaction in the data stream by μ_J. Therefore, the expectation of the area of the itemset J in a window of size w is $\mu_J.|J|.w$.

Given a window $W_w = T_1 \cdots T_{w-1}T_w$, assume that J is kept in the candidate list C_i of transaction T_i. The *lower-bound* J^- is calculated as in the *cTile* algorithm. Instead of using an upper bound J^+ on the area of J, we compute an *estimate* of the area of J denoted by J^*.

Since μ_J is unknown, we cannot directly define J^* as $\mu_J.|J|.w$. The probability μ_J is thus estimated in the sub-window $T_i \cdots T_w$. However, if this sub-window is too small, the estimation of μ_J is not accurate. Therefore, we introduced a threshold $L \leq w$: if the sub-window is smaller than L, we do not use this estimation and fall back on using the upper bound J^+ for J^*:

$$J^* = \begin{cases} J^- + (i-1) \times |J| & \text{if } w - i + 1 \leq L \text{ (i.e., } J^* = J^+) \\ \frac{J^-}{(w-i+1)} \times w & \text{if } w - i + 1 > L \end{cases} \quad (2)$$

All the steps of the of the *aTile* algorithm are very similar to the *cTile* algorithm except that it uses J^* instead of J^+ for candidate pruning. In the experiment section we empirically show that the *aTile* algorithm is more efficient than the *cTile* algorithm because its candidate set is more concise. An important property of the *aTile* algorithm is that the error rate on the accuracy of the result is bounded if we assume that the transactions are i.i.d. in a window of size $2w$.

5 Theoretical Analysis

In this section, we show theoretical bounds on the probability of errors induced by the *aTile* algorithm. We mainly show that the error rate is extremely low when L is large enough. We consider two types of error event: *False negative (FN)*: in a window W_{t^*} for some t^*, there is a true top-k largest tile that is not present in the results returned by the *aTile* algorithm and *False positive (FP)*: in a window W_{t^*} for some t^*, there is a non top-k largest tile that is present in the result returned by the *aTile* algorithm.

5.1 False Negative Bound

Let us assume that I is a true top-k largest tile in the window W_{t^*} and $I_k^{t^*}$ is ranked at position k in the list of a true top-k largest tile of the window W_{t^*}. Let $\triangle \geq 0$ be the difference between the average area of I and $I_k^{t^*}$ in the window W_{t^*}, i.e. $\triangle = \frac{A_{t^*}(I) - A_{t^*}(I_k^{t^*})}{w}$. The following lemma show the relationship between the probability of a false negative, L and $\triangle$:

Lemma 1 *If the transactions are independent and identically distributed (i.i.d) in a window of size* $2w$*, the probability that a random top-*k *tile is not reported, is bounded as follows:*

$$Pr(FN) < 4w * e^{-\frac{L\triangle^2}{2|I|^2}}$$

Proof. A false negative happens in the window W_{t^*} with respect to an itemset I if its area is underestimated in that window. This only happens if there exists at least one moment $t < t^*$ such that I is pruned from the candidate list C_i $(t - w < i \leq t)$.

Given a time t, let W_0 be a window containing transactions $T_i, T_{i+1}, \cdots, T_t$ where $|W_0| = w_o > L$ and let $f_0(I)$ be the frequency of I in the window W_0. When t is given, the event "I is removed from C_i" (denoted by R_t) only happens when the estimate of the *upper-bound* on the area of I is less than the area of $I_k^{t^*}$, i.e. $f_{W_0}(I)|I|\frac{w}{w_0} < A_{t^*}(I_k^{t^*})$. Therefore, we have:

$$\begin{aligned}
Pr(R_t) &< Pr\left(f_0(I)|I|\frac{w}{w_0} \leq A_{t^*}(I_k^{t^*})\right) < Pr\left(f_0(I)|I|\frac{w}{w_0} < A_{t^*}(I) - w\triangle\right) \\
&< Pr\left(\frac{f_0(I)}{w_0} < \frac{f_{t^*}(I)}{w} - \frac{\triangle}{|I|}\right) < Pr\left(\left|\frac{f_0(I)}{w_0} - \frac{f_{t^*}(I)}{w}\right| > \frac{\triangle}{|I|}\right) \\
&< Pr\left(\left|\frac{f_0(I)}{w_0} - \mu_I\right| + \left|\frac{f_{t^*}(I)}{w} - \mu_I\right| > \frac{\triangle}{|I|}\right) \\
&< Pr\left(\left|\frac{f_0(I)}{w_0} - \mu_I\right| > \frac{\triangle_0}{w_0}\right) + Pr\left(\left|\frac{f_{t^*}(I)}{w} - \mu_I\right| > \frac{\triangle_1}{w}\right) \\
&< Pr\left(|f_0(I) - \mu_I w_0| > \triangle_0\right) + Pr\left(|f_{t^*}(I) - \mu_I w| > \triangle_1\right)
\end{aligned}$$

Where $\triangle_0 = \frac{\triangle w_0}{2|I|}$ and $\triangle_1 = \frac{w\triangle}{2|I|}$. It is important to notice that $w_0\mu_I$ and $w\mu_I$ are the expectation of the frequency of I in the window W_0 and the window W_{t^*} respectively. Since the transactions are independent to each other, according to the Hoeffding inequality [11] we have:

$$Pr\left(|f_0(I) - w_0\mu_I| > \triangle_0\right) < 2e^{-\frac{2\triangle_0^2}{w_0}} < 2e^{-w_0\frac{\triangle^2}{2|I|^2}} < 2e^{-L\frac{\triangle^2}{2|I|^2}} \tag{3}$$

A similar inequality can be obtained for bounding the second term as follows:

$$Pr\left(|f_{t^*}(I) - w\mu_I| > \triangle_1\right) < 2e^{-\frac{2\triangle_1^2}{w}} < 2e^{-w\frac{\triangle^2}{2|I|^2}} < 2e^{-L\frac{\triangle^2}{2|I|^2}} \tag{4}$$

Moreover, since FN happens only when there at least one moment t such that R_t happens, therefore:

$$Pr(FN) < Pr(\cup_{t^*-w<t\leq t^*} R_t) < \sum_{t^*-w<t\leq t^*} Pr(R_t) \tag{5}$$

Inequalities 3, 4 and 5 prove the lemma.

A direct corollary of Lemma 1 is shown in the following theorem:

Theorem 2 *If I is strictly more important than $I_k^{t^*}$, i.e. the expectation of the area of I in a transaction is strictly greater than the expectation of the area of $I_k^{t^*}$ in a transaction and $L = O(w)$ then:* $\lim_{L \longrightarrow \infty} Pr(FN) = 0$

Proof. Let $E(A(I))$ be the expectation of the area of I in a transaction and $E(A(I_k^{t^*}))$ be the expectation of the area of $I_k^{t^*}$ in a transaction. Since $\triangle = \frac{A_{t^*}(I) - A_{t^*}(I_k^{t^*})}{w}$ we can imply that:

$$\frac{\triangle}{|I|} = \frac{A_{t^*}(I) - A_{t^*}(I_k^{t^*})}{w|I|} \simeq \frac{E(A(I)) - E(A(I_k^{t^*}))}{|I|} \tag{6}$$

The last equation is the result of the law of large number when L goes to ∞. From the last equation we can imply that $\lim_{L \longrightarrow \infty} 4we^{-L\frac{\triangle^2}{2|I|^2}} = 0$ from which the theorem is proved.

5.2 False Positive Bound

In this subsection we prove bound for false positive error which can be obtained in a similar way as the bound of false negative.

Let J be an itemset that is not a true top-k largest tile in the window W_{t^*} but returned by the $aTile$ algorithm as a false positive tile. An itemset ranked at position k in the list of a true top-k largest tile of the window W_{t^*} is denoted by $I_k^{t^*}$. Let $\triangle \geq 0$ be the difference between the area of J and $I_k^{t^*}$ in the window W_{t^*}, i.e. $\triangle = \frac{A_{t^*}(I_k^{t^*}) - A_{t^*}(J)}{w}$.

The following lemma show the relationship between the probability of a false positive and L, $\triangle$ (proof is similar to Lemma 1):

Lemma 2 *If the transactions are i.i.d. in a window of size $2w$ the probability of false positive, i.e. the event that a random non top-*k *tile is reported, is bounded as follows:*

$$Pr(FP) < 4w * e^{-\frac{L\triangle^2}{2|I|^2}}$$

A corollary of Lemma 2 is shown as follows (proof is similar to Theorem 2):

Theorem 3 *If tile size is bounded and if J is strictly less important than $I_k^{t^*}$, i.e. the expectation of the area of J is strictly less than the expectation of the area of $I_k^{t^*}$ in a transaction and $L = O(w)$ then:* $\lim_{L \longrightarrow \infty} Pr(FP) = 0$

Theorem 3 and Theorem 2 show an interesting result that the probability of false positive and false negative decrease exponentially with $L = O(w)$. It is important to notice that the condition $L = O(w)$ can be replaced by a weaker assumption $L = k \log w$ for some constant values k. If k is large enough the bound is also closed to zero. Although the bounds are not tight, in experiments, we empirically show that false negative rate and false positive rate are negligible even when L is set to a small value.

5.3 Long Lasting Tiles

The two previous Theorems give probabilistic results. We can also show a deterministic one: if an itemset stays in the top-k largest tiles for more than w consecutive windows and its area is at least twice the area of the k-th tile, then the *aTile* algorithm finds it. An important point of this theorem is that it does not depends on the value of L. Therefore, even if the probability of false negative or false positive is higher with a small L, the algorithm is still able to mine these tiles that we call the long lasting tiles.

Theorem 4 *Let z be a time and J be an itemset. If the area of J in every windows W_t with $z \leq t < z+w$ is larger than two time the area of the k-th largest tile in this window (i.e., $A_t(J) \geq 2A_t(I_k^t)$), then there is a time $z-w < t^* < z+w$ such that J (or one of its supersets) is in C_{t^*} and is never pruned by* aTile.

Proof. We define hypothesis H as: t^* *does not exist*. We will show that if H is true, it leads to a contradiction and thus the theorem must be true. We denote by $[x, y]$ the subsequence of transactions $T_x \cdots T_y$. The occurrence times of J in the subsequence $[z-w+1, z+w-1]$ are denoted by $z-w < t_1 < \ldots < t_k < z+w$.

If H is true, then for every t_i, itemset J and its supersets must be pruned from C_{t_i} at some time t'_i no later than $t_i + w - 1$, i.e., $t'_i < t_i + w$.

Let S_i be the subsequence $[t_i, t'_i] = T_{t_i} \cdots T_{t'_i}$. If for all $z - w < t_i \leq z$, we have $t'_i \leq z$ then we define $W_0 = W_z$. Otherwise, we take $W_0 = W_{t'_{max}}$ where $t'_{max} = \max\{t'_i \mid t_i \leq z\}$.

We now construct a set $\mathcal{S}$ of these subsequences S_i such that every subsequence of $\mathcal{S}$ is included in W_0, every occurrence of J is in at least one of these subsequences, and at most two of these subsequences intersect at any given time. We start with $\mathcal{S} = \emptyset$ if $W_0 = W_z$ or with $\mathcal{S} = \{[t_{max}, t'_{max}]\}$ otherwise. We scan the window W_0 from left to right. If there is an occurrence of J at time t_i not already in a subsequence of S, then we add S_i in $\mathcal{S}$. All the added subsequences are disjoint by construction, only the last added one may intersect with (t_{max}, t'_{max}).

For every $S_i \in \mathcal{S}$, let A_i be the area of the k-th largest tile of S_i used to prune J, i.e., $J^* < A_i$. Since by Eq. 2 $J^* \geq wA_{S_i}(J)/(t'_i - t_i + 1)$, we have $wA_{S_i}(J)/(t'_i - t_i + 1) < A_i$. If we define $A_m = \max A_i$, then $A_{S_i}(J) < A_m(t'_i - t_i + 1)/w$ and by summation for all $S_i \in \mathcal{S}$: $\sum_i A_{S_i}(J) < A_m \sum_i (t'_i - t_i + 1)/w$. Since all occurrences of J are covered by at least one S_i, $\sum_i A_{S_i}(J) > A_{W_0}(J)$ and since at most two subsequences S_i from $\mathcal{S}$ intersect at any given time, $\sum_i (t'_i - t_i + 1) \leq 2w$ and thus $A_{W_0}(J) < 2A_m$. Since A_m is the size of the k-th tile of one of the $S_i \subseteq W_0$, it is less than the size of the k-th tile in the whole window W_0. Finally, $A_{W_0}(J)$ is strictly less than two time the area of the k-th tile in W_0 which is a contradiction.

6 Experiments

In this section, we perform experiments with two real-life data streams to compare the proposed algorithms to the baseline approaches with respect to the efficiency and the accuracy of the results. The two datasets are:

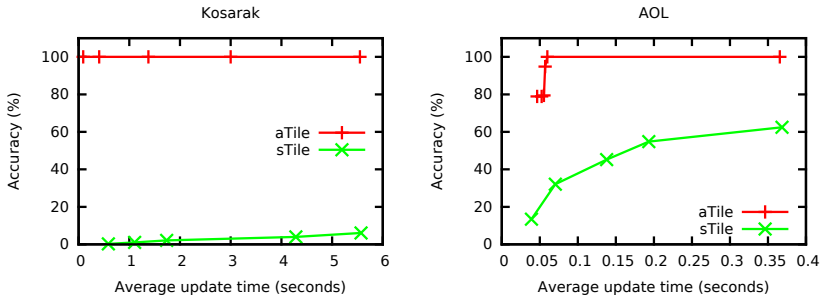
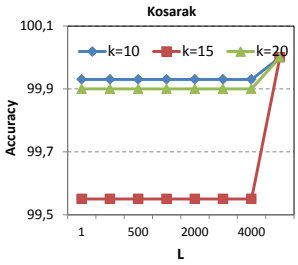
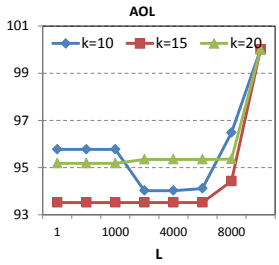


Fig. 3. Accuracy versus average update time in seconds of the *aTile* algorithm and the *sTile* algorithm when L and the number samples increases





Kosarak		AOL	
w	Accuracy	w	Accuracy
500	99.1	1000	93,4
1000	99.3	2000	95,2
2000	98.3	4000	89,7
3000	99.5	6000	85,1
4000	99.8	8000	89,4
5000	99.9	10000	95,7

Fig. 4. The average accuracy of the *aTile* algorithm when L is varied. In the table: the average accuracy of the top-*10* largest tiles returned by *aTile* when the window size is varied and $L = 0.1w$

– Kosarak: 1M transactions of click log by the users of a website. Item are a pages in the website. The largest transaction contains 500 items.
– AOL: 100K search queries by the users of the AOL search engine. Each item is a keyword in the search query. Most queries are short and the longest query contains only 26 keywords.

The datasets and the source codes of *aTile* and *cTile* in C++ are available for download[5]. The baseline algorithms for comparison are as follows:

– *Tile*: the original implementation of the tiling algorithm for static database [7]. In order to adopt this algorithm for a data stream, we use *Tile* to recalculate the top-k largest tiles from scratch whenever the window is sliding.
– *sTile*: a sampling based technique proposed in [2]. The *sTile* algorithm samples N itemsets from the window such that each itemset is sampled with probability proportional to the area of the itemset. The top-k largest tiles are extracted from the samples every time the window is sliding.

[5] `http://www.win.tue.nl/~lamthuy/tile.htm`

sTile		aTile	
# samples	Sum of area	L	Sum of area
5000	2652	1000	20384
10000	2753	2000	20384
20000	2822	3000	20384
40000	2918	4000	20384
80000	2992	5000	20388

Fig. 5. The average sum (larger is better) of the area of the top-10 tiles returned by the *aTile* and the *sTile* algorithms.

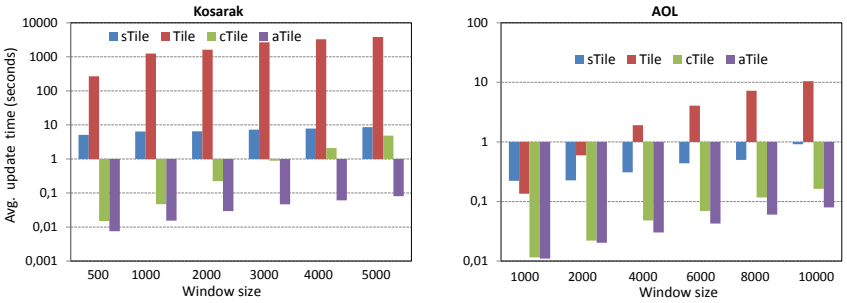


Fig. 6. Average time per update according to the window size.

6.1 Accuracy

We performed experiments to demonstrate the effectiveness of the *aTile* algorithm in term of accuracy. In the first experiment the window size is set to $w = 5000$ for the Kosarak dataset and $w = 10000$ for the AOL dataset. Figure 4 shows the average accuracy calculated as the precision of the top-k largest tiles returned by the *aTile* algorithm when L is varied. Even for very small L, the accuracy is very high, e.g. 99% in the Kosarak dataset and 93% in the AOL dataset. The accuracy increases and reaches 100% accuracy when L is increased. In the same figure, we can also see that the results are similar when k is varied.

In order to compare to the baseline algorithm *sTile*, we plot the accuracy (y-axis) versus average update time (x-axis) in Figure 3. We varied both L for the *aTile* algorithm and the number of samples of the *sTile* algorithm. Recall that the *sTile* algorithm depends on the number of samples it collects from the window. When the number of samples increases, so do the accuracy and the update time. This fact is illustrated in Figure 3 in which the *sTile* algorithm is significantly less accurate than the *aTile* algorithm given about the same average update time.

The accuracy of *sTile* may be negatively influenced by the fact that *sTile* has a very low probability to find each top-k tile. When we calculated the average sum of the area of the top-*10* tiles in the Kosarak dataset for $w = 5000$, we observed that it was significantly lower for *sTile* than for the *aTile* algorithm (Figure 5). This confirmed that *sTile* is not able to find the large tiles with a reasonable number of samples.

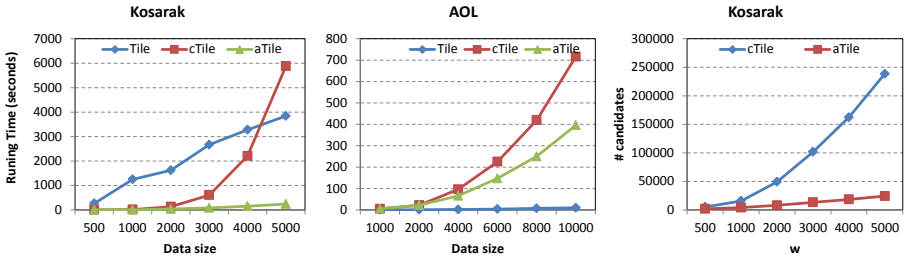


Fig. 7. Running time according to the database size when the algorithm is used to mine large tiles in a database instead of a stream. The right most subplot shows the number of candidates in the summary when the window size increases.

6.2 Efficiency

Figure 6 shows average update time of four algorithms for $k = 10$ and different values of w. The number of samples in the *sTile* algorithm was set to $N = 80000$ (when N is larger, the algorithm become significantly slower). We set $L = \frac{w}{10}$ (as shown in Sect. 6.1, the accuracy is then always above 85%). In term of update time, *aTile* is up to an order of magnitude (50x) faster than the *cTile* algorithm and about two to three orders of magnitude faster than the *sTile* (100x) and the *Tile* (1000x) algorithms. The speed-up increases with the window size.

In Figure 7, the last plot shows the number of candidates kept in the summary of the *aTile* and the *cTile* algorithm for varying values of w (the result of the AOL dataset is omitted because it is very similar to the results of the Kosarak dataset in this experiment). The *aTile* algorithm is more memory efficient than the *cTile* algorithm as the number of candidates it keeps in the summary is much lower.

Finally, in Figure 7 we show the running time when the *aTile*, *cTile* and *Tile* algorithms are used to find the top-*10* largest tiles in a static corpus with varying size. The purpose of the experiment is to see whether the *aTile* and the *cTile* algorithm can find the large tiles in a static corpus more efficiently than the *Tile* algorithm. For the AOL dataset, when most of transactions are small, the *Tile* algorithm is the fastest one. However, for the Kosarak dataset, when the average transaction size is larger, *aTile* outperforms both *Tile* and *cTile*. Therefore, the *aTile* algorithm can not only be used for mining large tiles from a data stream but also to mine large tiles from a database efficiently.

6.3 Application to Topics Monitoring in the AOL Query Stream

In order to show a potential application of the work we created a demo video[6] to visualize the top-k largest tiles of the AOL query stream. Each snapshot of the video corresponds to a list of the largest tiles extracted from a sliding window.

[6] `http://www.youtube.com/watch?v=3UCjs9d91_g`

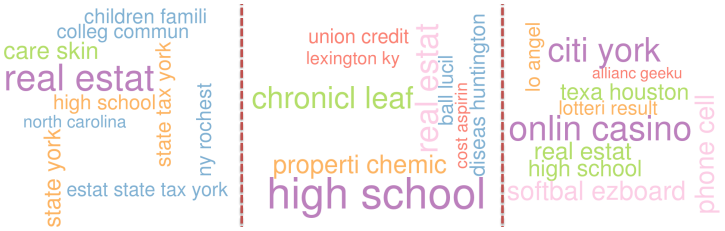


Fig. 8. Top-10 largest tiles in windows (ordered by timestamps) from the AOL query stream. Larger words correspond to tiles with bigger area. The tile "high school" emerges in the second window related to searches for the "high school musical movies".

Figure 8 shows example snapshots taken from the demo video. Three snapshots of the demo (ordered by timestamps) show the evolution of the largest tiles overtime. Each snapshot is visualized by the *wordcloud* tool in *R*. Larger words correspond to tiles with larger area.

For example, in the first snapshot the keyword *"real estate"* is the most important tile. However, it becomes less important in the second and the third snapshot. Meanwhile, new important tiles such as *"high school"* is emerging as search for the "high school musical movies" increases in 2006. With this demo, users can track the dynamic of important search topics online.

6.4 Application to Tracking in Videos

We investigate how mining top-k largest tiles in a data stream can be useful for analysis of videos and, in particular, for tracking. A more detailed description of this application can be found in [3]. We worked on a real video made of 5619 frames. This video is shot from a car while following another car (the main object). The main object is present in almost all the frames of the video. If we are able to represent the video as a data stream and use our tile mining algorithm on it, it should discover the car as a large tile or a set of large tiles.

We used the segmentation algorithm[7] [8] to generate a stream of graphs (one graph per frame). Each graph is a Region Adjacency Graph (RAG) where each node is a region (a set of adjacent pixels) and two nodes are connected if their regions are adjacent. This algorithm performs a temporal segmentation, which means that a given region (and thus, the corresponding node) is present in several successive frames. However, a single region is not enough to track an object in the video: due to change in the object pose or illumination, some regions will split, or merge or disappear and so do the corresponding nodes. We then build a transaction consisting in the set of nodes of each RAG.

Mining tiles on this data stream results in tiles containing regions spread all over the frames. Moreover, the top-k tiles are generally very similar (only differing by one or two regions). Indeed, by removing the information about the

[7] http://www.videosegmentation.com

Fig. 9. Frame of the video.

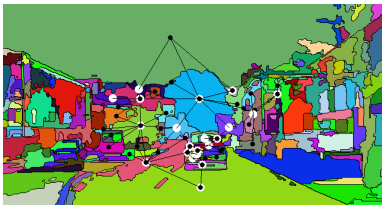


Fig. 10. top-30 star-shaped tiles in the segmented frame.

adjacency of the regions, it was not possible to find meaningful tiles. The main problem is that the regions composing the car (or more generally, an object to be tracked) must be "close" to each other and this was not taken into account in the original setting. Therefore, we added a spatial constraint to the problem. We chose to mine tiles that are star subgraphs of the RAGs. A star subgraph is a graph with nodes $i_0, i_1, \ldots i_n$ where $i_1, \ldots, i_n$ are nodes adjacent to i_0 in the RAG. This formalization ensures that the items in a tile are adjacent to a "center" item i_0. Moreover, it is easy to integrate this constraint into our algorithms: the intersection of two star graphs with the same center is just the intersection of their set of nodes. The algorithm is otherwise unchanged.

We extracted the top-30 tiles with a window of 300 frames (see an example on Fig 10). Then we checked if the extracted tiles could be useful to track the main object (car). We manually drawn the bounding box containing the car in each fifth frame. Then, we selected, for each frame, the tile with the best precision. We end up with a set of 69 tiles with an overall precision of 0.92 and recall of 0.74 on the whole video (and even a recall of 1 on the 2000 first frames). This means that the tiles can actually be used as a high level feature for tracking. Of course, in a real application, the bounding box is not known. But techniques like those described in [5] can be used to build tracks from the extracted tiles.

7 Conclusions and Future Works

In this paper, we proposed two algorithms for mining the top-k largest tiles from a data stream with a sliding window model. The first candidate-based algorithm *cTile* solves the problem exactly but the update time becomes important when the window size increases. The second one is an approximation algorithm with theoretical bounds on the error rate. Experiments with two real-life datasets show that the approximation algorithm can find large tiles with high accuracy while being an order of magnitude faster than the candidate based algorithm and two to three order of magnitude faster than the other baseline algorithms.

We also show potential applications of large tiles mining in monitoring emerging popular topics in a search engine query log or in a tracking problem. A possible extension for future work is to consider mining different types of tiles with constraints for meaningful applications.

8 Acknowledgements

This work is supported by the Netherlands Organization for Scientific Research (NWO) through the project *Mining Complex Patterns in Stream* (COMPASS) and the French ANR project SoLStiCe (ANR-13-BS02-0002-01). We would like to thank Dr. Bart Goethals for providing us the implementation of the *Tile* algorithm and Pr. Toon Calders for his precious comments.

References

1. Aggarwal, C.C. (ed.): Data Streams - Models and Algorithms, Advances in Database Systems, vol. 31. Springer (2007)
2. Boley, M., Lucchese, C., Paurat, D., Gärtner, T.: Direct local pattern sampling by efficient two-step random procedures. In: KDD. pp. 582–590 (2011)
3. Calders, T., Fromont, É., Jeudy, B., Lam, H.T.: Analysis of videos using tile mining. In: Real-World Challenges for Data Stream Mining Workshop (2013)
4. Cerf, L., Besson, J., Nguyen, K.N., Boulicaut, J.F.: Closed and noise-tolerant patterns in n-ary relations. Data Min. Knowl. Discov. 26(3), 574–619 (2013)
5. Diot, F., Fromont, É., Jeudy, B., Marilly, E., Martinot, O.: Graph mining for object tracking in videos. In: ECML/PKDD (1). LNCS, vol. 7523, pp. 394–409 (2012)
6. Fernando, B., Fromont, É., Tuytelaars, T.: Effective use of frequent itemset mining for image classification. In: ECCV (1). pp. 214–227 (2012)
7. Geerts, F., Goethals, B., Mielikäinen, T.: Tiling databases. In: Discovery Science. pp. 278–289 (2004)
8. Grundmann, M., Kwatra, V., Han, M., Essa, I.: Efficient hierarchical graph-based video segmentation. CVPR (2010)
9. van Leeuwen, M., Knobbe, A.J.: Diverse subgroup set discovery. Data Min. Knowl. Discov. 25(2), 208–242 (2012)
10. Lu, H., Vaidya, J., Atluri, V.: Optimal boolean matrix decomposition: Application to role engineering. In: ICDE. pp. 297–306 (2008)
11. Motwani, R., Raghavan, P.: Randomized algorithms. Cambridge University Press, USA (1995)
12. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. pp. 398–416. ICDT, Springer-Verlag (1999)
13. Remmerie, N., Vijlder, T.D., Valkenborg, D., Laukens, K., Smets, K., Vreeken, J., Mertens, I., Carpentier, S.C., Panis, B., Jaeger, G.D., Blust, R., Prinsen, E., Witters, E.: Unraveling tobacco by-2 protein complexes with {BN} page/lcms/ms and clustering methods. Journal of Proteomics 74(8), 1201 – 1217 (2011)
14. Smets, K., Vreeken, J.: The odd one out: Identifying and characterising anomalies. In: SDM. pp. 804–815 (2011)
15. Tatti, N., Vreeken, J.: Discovering descriptive tile trees by mining optimal geometric subtiles. In: ECML/PKDD (1). pp. 9–24 (2012)
16. Vaidya, J., Atluri, V., Guo, Q.: The role mining problem: A formal perspective. ACM Trans. Inf. Syst. Secur. 13(3) (2010)
17. Vreeken, J., van Leeuwen, M., Siebes, A.: Krimp: mining itemsets that compress. Data Min. Knowl. Discov. 23(1), 169–214 (2011)